

# Package ‘etl’

June 2, 2020

**Type** Package

**Title** Extract-Transform-Load Framework for Medium Data

**Version** 0.3.9

**Maintainer** Ben Baumer <ben.baumer@gmail.com>

**Description** A predictable and pipeable framework for performing ETL (extract-transform-load) operations on publicly-accessible medium-sized data set. This package sets up the method structure and implements generic functions. Packages that depend on this package download specific data sets from the Internet, clean them up, and import them into a local or remote relational database management system.

**License** CC0

**LazyData** TRUE

**Imports** DBI, datasets, downloader, fs, janitor, lubridate, methods, stringr, readr, rlang, rvest, tibble, usethis, utils, xml2

**Depends** R (>= 2.10), dplyr

**Suggests** dbplyr, knitr, RSQLite, RPostgreSQL, RMySQL, ggplot2, testthat, rmarkdown

**URL** <http://github.com/beanumber/etl>

**BugReports** <https://github.com/beanumber/etl/issues>

**RoxygenNote** 7.1.0

**Encoding** UTF-8

**VignetteBuilder** knitr, rmarkdown, ggplot2, dplyr, dbplyr

**NeedsCompilation** no

**Author** Ben Baumer [aut, cre],  
Carson Sievert [ctb],  
Natalia Iannucci [ctb]

**Repository** CRAN

**Date/Publication** 2020-06-02 15:50:06 UTC

## R topics documented:

create_etl_package . . . . .	2
dbRunScript . . . . .	3
dbWipe . . . . .	4
db_type . . . . .	4
etl . . . . .	5
etl_cleanup . . . . .	7
etl_init . . . . .	10
match_files_by_year_months . . . . .	12
smart_download . . . . .	13
smart_upload . . . . .	14
src_mysql_cnf . . . . .	14
valid_year_month . . . . .	15
<b>Index</b>	<b>17</b>

---

create_etl_package	<i>Create an ETL package skeleton</i>
--------------------	---------------------------------------

---

### Description

Create an ETL package skeleton

### Usage

```
create_etl_package(...)
```

### Arguments

... arguments passed to [create\\_package](#)

### Details

Extends [create\\_package](#) and places a template source file in the R subdirectory of the new package. The file has a working stub of `etl_extract`. The new package can be built immediately and run.

New S3 methods for [etl\\_transform](#) and [etl\\_load](#) can be added if necessary, but the default methods may suffice.

### See Also

[etl\\_extract](#), [etl\\_transform](#), [etl\\_load](#)

**Examples**

```
## Not run:
path <- file.path(tempdir(), "scorecard")
create_etl_package(path)

## End(Not run)
# Now switch projects, and "Install and Restart"
```

---

dbRunScript	<i>Execute an SQL script</i>
-------------	------------------------------

---

**Description**

Execute an SQL script

**Usage**

```
dbRunScript(conn, script, echo = FALSE, ...)
```

**Arguments**

conn	a <a href="#">DBIConnection-class</a> object
script	Either a filename pointing to an SQL script or a character vector of length 1 containing SQL.
echo	print the SQL commands to the output?
...	arguments passed to <a href="#">dbExecute</a>

**Details**

The SQL script file must be ; delimited.

**Value**

a list of results from dbExecute for each of the individual SQL statements in script.

**Examples**

```
sql <- "SHOW TABLES; SELECT 1+1 as Two;"
sql2 <- system.file("sql", "mtcars.mysql", package = "etl")
sql3 <- "SELECT * FROM user WHERE user = 'mysql';SELECT * FROM user WHERE 't' = 't';"

if (require(RSQLite)) {
  con <- dbConnect(RSQLite::SQLite())
  dbRunScript(con, "SELECT 1+1 as Two; VACUUM; ANALYZE;")
}
## Not run:
if (require(RMySQL)) {
  con <- dbConnect(RMySQL::MySQL(), default.file = path.expand("~/my.cnf"),
```

```

    group = "client", user = NULL, password = NULL, dbname = "mysql", host = "127.0.0.1")
  dbRunScript(con, script = sql1)
  dbRunScript(con, script = sql2)
  dbRunScript(con, script = sql3)
  dbDisconnect(con)
}

## End(Not run)

```

---

dbWipe	<i>Wipe out all tables in a database</i>
--------	--

---

### Description

Wipe out all tables in a database

### Usage

```
dbWipe(conn, ...)
```

### Arguments

conn	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
...	Other parameters passed on to methods.

### Details

Finds all tables within a database and removes them

---

db_type	<i>Return the database type for an ETL or DBI connection</i>
---------	--

---

### Description

Return the database type for an ETL or DBI connection

### Usage

```

db_type(obj, ...)

## S3 method for class 'src_dbi'
db_type(obj, ...)

## S3 method for class 'DBIConnection'
db_type(obj, ...)

```

**Arguments**

obj                   and `etl` or `DBIConnection-class` object  
 ...                   currently ignored

**Examples**

```
if (require(RMySQL) && mysqlHasDefault()) {
  # connect to test database using rs-dbi
  db <- src_mysql_cnf()
  class(db)
  db
  # connect to another server using the 'client' group
  db_type(db)
  db_type(db$con)
}
```

---

etl	<i>Initialize an etl object</i>
-----	---------------------------------

---

**Description**

Initialize an etl object

**Usage**

```
etl(x, db = NULL, dir = tempdir(), ...)

## Default S3 method:
etl(x, db = NULL, dir = tempdir(), ...)

## S3 method for class 'etl'
summary(object, ...)

is.etl(object)

## S3 method for class 'etl'
print(x, ...)
```

**Arguments**

x                   the name of the etl package that you wish to populate with data. This determines the class of the resulting `etl` object, which determines method dispatch of `etl_*`() functions. There is no default, but you can use `mtcars` as an test example.

db                   a database connection that inherits from `src_dbi`. It is `NULL` by default, which results in a `src_sqlite` connection being created in `dir`.

dir                  a directory to store the raw and processed data files

... arguments passed to methods (currently ignored)  
 object an object for which a summary is desired.

### Details

A constructor function that instantiates an `etl` object. An `etl` object extends a `src_dbi` object. It also has attributes for:

**pkg** the name of the `etl` package corresponding to the data source

**dir** the directory where the raw and processed data are stored

**raw\_dir** the directory where the raw data files are stored

**load\_dir** the directory where the processed data files are stored

Just like any `src_dbi` object, an `etl` object is a data source backed by an SQL database. However, an `etl` object has additional functionality based on the presumption that the SQL database will be populated from data files stored on the local hard disk. The ETL functions documented in `etl_create` provide the necessary functionality for **extracting** data from the Internet to `raw_dir`, **transforming** those data and placing the cleaned up data (usually in CSV format) into `load_dir`, and finally **loading** the clean data into the SQL database.

### Value

For `etl`, an object of class `etl_x` and `etl` that inherits from `src_dbi`

For `is.etl`, TRUE or FALSE, depending on whether `x` has class `etl`

### See Also

[etl\\_create](#)

### Examples

```
# Instantiate the etl object
cars <- etl("mtcars")
str(cars)
is.etl(cars)
summary(cars)

## Not run:
# connect to a PostgreSQL server
if (require(RPostgreSQL)) {
  db <- src_postgres("mtcars", user = "postgres", host = "localhost")
  cars <- etl("mtcars", db)
}

## End(Not run)

# Do it step-by-step
cars %>%
  etl_extract() %>%
```

```

    etl_transform() %>%
    etl_load()
src_tbls(cars)
cars %>%
  tbl("mtcars") %>%
  group_by(cyl) %>%
  summarize(N = n(), mean_mpg = mean(mpg))

# Do it all in one step
cars2 <- etl("mtcars")
cars2 %>%
  etl_update()
src_tbls(cars2)

# generic summary function provides information about the object
cars <- etl("mtcars")
summary(cars)
cars <- etl("mtcars")
# returns TRUE
is.etl(cars)

# returns FALSE
is.etl("hello world")
cars <- etl("mtcars") %>%
  etl_create()
cars

```

---

etl\_cleanup

*ETL functions for working with medium sized data*


---

## Description

These generic functions provide a systematic approach for performing ETL (exchange-transform-load) operations on medium sized data.

## Usage

```

etl_cleanup(obj, ...)

## Default S3 method:
etl_cleanup(
  obj,
  delete_raw = FALSE,
  delete_load = FALSE,
  pattern = "\\.(csv|zip)$",
  ...
)

```

```
etl_create(obj, ...)

## Default S3 method:
etl_create(obj, ...)

etl_update(obj, ...)

## Default S3 method:
etl_update(obj, ...)

etl_extract(obj, ...)

## Default S3 method:
etl_extract(obj, ...)

## S3 method for class 'etl_mtcars'
etl_extract(obj, ...)

## S3 method for class 'etl_cities'
etl_extract(obj, ...)

etl_load(obj, ...)

## Default S3 method:
etl_load(obj, ...)

etl_transform(obj, ...)

## Default S3 method:
etl_transform(obj, ...)

## S3 method for class 'etl_cities'
etl_transform(obj, ...)
```

### Arguments

obj	an <a href="#">etl</a> object
...	arguments passed to methods
delete_raw	should files be deleted from the raw_dir?
delete_load	should files be deleted from the load_dir?
pattern	regular expression matching file names to be deleted. By default, this matches filenames ending in .csv and .zip.

### Details

The purposes of these functions are to download data from a particular data source from the Internet, process it, and load it into a SQL database server.

There are five primary functions:



**etl\_init** Initialize the database schema.

**etl\_extract** Download data from the Internet and store it locally in its raw form.

**etl\_transform** Manipulate the raw data such that it can be loaded into a database table. Usually, this means converting the raw data to (a series of) CSV files, which are also stored locally.

**etl\_load** Load the transformed data into the database.

**etl\_cleanup** Perform housekeeping, such as deleting unnecessary raw data files.

Additionally, two convenience functions chain these operations together:

**etl\_create** Run all five functions in succession. This is useful when you want to create the database from scratch.

**etl\_update** Run the `etl_extract-etl_transform-etl_load` functions in succession. This is useful when the database already exists, but you want to insert some new data.

## Value

Each one of these functions returns an `etl` object, invisibly.

## See Also

`etl`, `etl_init`

## Examples

```
## Not run:
if (require(RPostgreSQL)) {
  db <- src_postgres(dbname = "mtcars", user = "postgres", host = "localhost")
  cars <- etl("mtcars", db)
}
if (require(RMySQL) && mysqlHasDefault()) {
  db <- src_mysql(dbname = "mtcars", user = "r-user",
                 host = "localhost", password = "mypass")
  cars <- etl("mtcars", db)
}

## End(Not run)
cars <- etl("mtcars")
cars %>%
  etl_extract() %>%
  etl_transform() %>%
  etl_load() %>%
  etl_cleanup()
cars

cars %>%
  tbl(from = "mtcars") %>%
  group_by(cyl) %>%
  summarise(N = n(), mean_mpg = mean(mpg))

# do it all in one step, and peek at the SQL creation script
```

```

cars %>%
  etl_create(echo = TRUE)
# specify a directory for the data
## Not run:
cars <- etl("mtcars", dir = "~/dumps/mtcars/")
str(cars)

## End(Not run)
cars <- etl("mtcars")
# Do it step-by-step
cars %>%
  etl_extract() %>%
  etl_transform() %>%
  etl_load()

# Note the somewhat imprecise data types for the columns. These are the default.
tbl(cars, "mtcars")

# But you can also specify your own schema if you want
schema <- system.file("sql", "init.sqlite", package = "etl")
cars %>%
  etl_init(schema) %>%
  etl_load()

```

---

etl\_init

*Initialize a database using a defined schema*


---

## Description

Initialize a database using a defined schema

## Usage

```

etl_init(
  obj,
  script = NULL,
  schema_name = "init",
  pkg = attr(obj, "pkg"),
  ext = NULL,
  ...
)

```

## Default S3 method:

```

etl_init(
  obj,
  script = NULL,
  schema_name = "init",
  pkg = attr(obj, "pkg"),
  ext = NULL,

```

```

    ...
  )

  find_schema(obj, schema_name = "init", pkg = attr(obj, "pkg"), ext = NULL, ...)

```

### Arguments

obj	An <a href="#">etl</a> object
script	either a vector of SQL commands to be executed, or a file path as a character vector containing an SQL initialization script. If NULL (the default), then the appropriate built-in schema will be fetched by <a href="#">find_schema</a> , if it exists. Note that the flavor of SQL in this file must match the type of the source. That is, if your object is of type <a href="#">src_mysql</a> , then make sure that the schema you specify here is written in MySQL (and not PostgreSQL). Please note that SQL syntax is not, in general, completely portable. Use with caution, as this may clobber any existing data you have in an existing database.
schema_name	The name of the schema. Default is <code>init</code> .
pkg	The package defining the schema. Should be set in <a href="#">etl</a> .
ext	The file extension used for the SQL schema file. If NULL (the default) it be inferred from the <code>src_*</code> class of <code>con</code> . For example, if <code>con</code> has class <a href="#">src_sqlite</a> then <code>ext</code> will be <code>sqlite</code> .
...	Currently ignored

### Details

If the table definitions are at all non-trivial, you may wish to include a pre-defined table schema. This function will retrieve it.

### Examples

```

cars <- etl("mtcars")
cars %>%
  etl_init()
cars %>%
  etl_init(script = sql("CREATE TABLE IF NOT EXISTS mtcars_alt (id INTEGER);"))
cars %>%
  etl_init(schema_name = "init")
init_script <- find_schema(cars, schema_name = "init")
cars %>%
  etl_init(script = init_script, echo = TRUE)
src_tbls(cars)

cars <- etl("mtcars")
find_schema(cars)
find_schema(cars, "init", "etl")
find_schema(cars, "my_crazy_schema", "etl")

```

---

`match_files_by_year_months`*Match year and month vectors to filenames*

---

**Description**

Match year and month vectors to filenames

Extracts a date from filenames

**Usage**

```
match_files_by_year_months(  
  files,  
  pattern,  
  years = as.numeric(format(Sys.Date(), "%Y")),  
  months = 1:12,  
  ...  
)  
  
extract_date_from_filename(files, pattern, ...)
```

**Arguments**

<code>files</code>	a character vector of filenames
<code>pattern</code>	a regular expression to be passed to <a href="#">fast_strptime</a>
<code>years</code>	a numeric vector of years
<code>months</code>	a numeric vector of months
<code>...</code>	arguments passed to <a href="#">fast_strptime</a>

**Value**

a character vector of files that match the pattern, year, and month arguments

a vector of [POSIXct](#) dates matching the pattern

**Examples**

```
## Not run:  
if (require(airlines)) {  
  airlines <- etl("airlines", dir = "~/Data/airlines") %>%  
    etl_extract(year = 1987)  
  summary(airlines)  
  match_files_by_year_months(list.files(attr(airlines, "raw_dir")),  
    pattern = "On_Time_On_Time_Performance_%Y_%m.zip", year = 1987)  
}  
  
## End(Not run)
```

---

smart_download	<i>Download only those files that don't already exist</i>
----------------	---

---

## Description

Download only those files that don't already exist

## Usage

```
smart_download(obj, src, new_filenames = basename(src), clobber = FALSE, ...)
```

## Arguments

obj	an <a href="#">etl</a> object
src	a character vector of URLs that you want to download
new_filenames	an optional character vector of filenames for the new (local) files. Defaults to having the same filenames as those in <code>src</code> .
clobber	do you want to clobber any existing files?
...	arguments passed to <a href="#">download</a>

## Details

Downloads only those files in `src` that are not already present in the directory specified by the `raw_dir` attribute of `obj`.

## Author(s)

idiom courtesy of Hadley Wickham

## Examples

```
cars <- etl("mtcars")
urls <- c("https://raw.githubusercontent.com/beanumber/etl/master/etl.Rproj",
"https://www.reddit.com/robots.txt")
smart_download(cars, src = urls)
# won't download again if the files are already there
smart_download(cars, src = urls)
# use clobber to overwrite
smart_download(cars, src = urls, clobber = TRUE)
```

---

smart_upload	<i>Upload a list of files to the DB</i>
--------------	---

---

**Description**

Upload a list of files to the DB

**Usage**

```
smart_upload(obj, src = NULL, tablenames = NULL, ...)
```

**Arguments**

obj	An <a href="#">etl</a> object
src	a list of CSV files to upload. If NULL, will return all CSVs in the load directory
tablenames	a list the same length as src of tablenames in the database corresponding to each of the files in src. If NULL, will default to the same name as src, without paths or file extensions.
...	arguments passed to <a href="#">dbWriteTable</a>

**Examples**

```
## Not run:
if (require(RMySQL)) {
  # must have pre-existing database "fec"
  # if not, try
  system("mysql -e 'CREATE DATABASE IF NOT EXISTS fec;'")
  db <- src_mysql_cnf(dbname = "mtcars")
}

## End(Not run)
```

---

src_mysql_cnf	<i>Connect to local MySQL Server using ~/.my.cnf</i>
---------------	--

---

**Description**

Connect to local MySQL Server using ~/.my.cnf

**Usage**

```
src_mysql_cnf(dbname = "test", groups = "rs-dbi", ...)
```

**Arguments**

dbname	name of the local database you wish to connect to. Default is test, as in <a href="#">mysqlHasDefault</a> .
groups	section of ~/.my.cnf file. Default is rs-dbi as in <a href="#">mysqlHasDefault</a>
...	arguments passed to <a href="#">src_mysql</a>

**See Also**

[src\\_mysql](#), [mysqlHasDefault](#)

**Examples**

```
if (require(RMySQL) && mysqlHasDefault()) {
  # connect to test database using rs-dbi
  db <- src_mysql_cnf()
  class(db)
  db
  # connect to another server using the 'client' group
  src_mysql_cnf(groups = "client")
}
```

---

valid_year_month	<i>Ensure that years and months are within a certain time span</i>
------------------	--

---

**Description**

Ensure that years and months are within a certain time span

**Usage**

```
valid_year_month(years, months, begin = "1870-01-01", end = Sys.Date())
```

**Arguments**

years	a numeric vector of years
months	a numeric vector of months
begin	the earliest valid date, defaults to the UNIX epoch
end	the most recent valid date, defaults to today

**Details**

Often, a data source will begin and end at known points in time. At the same time, many data sources are divided into monthly archives. Given a set of years and months, any combination of which should be considered valid, this function will return a [data.frame](#) in which each row is one of those valid year-month pairs. Further, if the optional begin and end arguments are specified, the rows will be filter to lie within that time interval. Furthermore, the first and last day of each month are computed.

**Value**

a `data.frame` with four variables: `year`, `month`, `month_begin` (the first day of the month), and `month_end` (the last day of the month).

**Examples**

```
valid_year_month(years = 1999:2001, months = c(1:3, 7))

# Mets in the World Series since the UNIX epoch
mets_ws <- c(1969, 1973, 1986, 2000, 2015)
valid_year_month(years = mets_ws, months = 10)

# Mets in the World Series during the Clinton administration
if (require(ggplot2)) {
  clinton <- filter(presidential, name == "Clinton")
  valid_year_month(years = mets_ws, months = 10,
    begin = clinton$start, end = clinton$end)
}
```



# Index

create\_etl\_package, 2  
create\_package, 2

data.frame, 15, 16  
db\_type, 4  
dbConnect(), 4  
dbExecute, 3  
DBIConnection, 4  
dbRunScript, 3  
dbWipe, 4  
dbWriteTable, 14  
download, 13

etl, 5, 5, 6, 8, 9, 11, 13, 14  
etl\_cleanup, 7  
etl\_create, 6  
etl\_create(etl\_cleanup), 7  
etl\_extract, 2  
etl\_extract(etl\_cleanup), 7  
etl\_init, 9, 10  
etl\_load, 2  
etl\_load(etl\_cleanup), 7  
etl\_transform, 2  
etl\_transform(etl\_cleanup), 7  
etl\_update(etl\_cleanup), 7  
extract\_date\_from\_filename  
    (match\_files\_by\_year\_months),  
    12

fast\_strptime, 12  
find\_schema, 11  
find\_schema(etl\_init), 10

is.etl, 6  
is.etl(etl), 5

match\_files\_by\_year\_months, 12  
mysqlHasDefault, 15

POSIXct, 12  
print.etl(etl), 5

smart\_download, 13  
smart\_upload, 14  
src\_dbi, 5, 6  
src\_mysql, 11, 15  
src\_mysql\_cnf, 14  
src\_sqlite, 5, 11  
summary.etl(etl), 5

valid\_year\_month, 15